Postel

7672

Rcvd 12Oct71

RFC 242
NIC 7672
Categories: D.4, D.7

DATA DESCRIPTIVE LANGUAGE FOR SHARED DATA

L. Haibt

A. Mullery

Thomas J. Watson Research Center

Yorktown Heights, N.Y.

July 19, 1971

Introduction

A primary consequence of the use of networks of computers is the demand for more efficient shared use of data.

Many of the impedements to easily shared data follow from the many diverse ways of representing and making reference to the same data. Almost all of these problems have been known before data was shared through computer networks, but the network facility has simply emphasized the problems.

For convenience of discussion, representation differences will be classified in three categories. The first category is one of very local representation - the bit patterns for the character set, for fixed point and floating point numbers. These differences are usually imposed by

differences in CPU's and storage devices. Translations from one representation to another at this level can usually be made a unit at a time (e.g. computer word by computer word) with the most serious problems occurring when there are some values in one representation scheme which have no corresponding meaning in the other representation scheme, as, for example, when trying to translate eight-bit bytes to six-bit bytes.

A second category of differences has to do with the representation of collections of data, e.g., their size, ordering and location.

A third category of representation differences which is a little difficult to characterize has to do with all the more complex structures that data collections may have - for example, files with indexes, fields with internal pointers and cross references, and collections of files such as partitioned data sets and generation data sets in OS 360.

The approach to coping with these problems within our project of Network/440 has been to work on the development of a descriptive language which would permit the specification of those aspects of data representation which would be subject to transformation in moving data about in a network. Then, the network data management system would be able to refer to the descriptions as needed in the data management function. For example, to a large extent, one could supply two descriptions to the data manager, one which indicates how data is now represented, and one which indicates how a copy of it should look, and the data management system could invoke the necessary transformations to make the proper copy.

This approach to specifying data transformation contrasts somewhat with systems, such as the RAND Form Machine, which provide a formalism for specifying the particular translation algorithms for changing from one form to another. The descriptor-to-descriptor approach seems to simplify the programming burden when creating new field formats. Neither method of specifying translations precludes the use of a Network Standard Representation.

Structure

The descriptive language assumes that data may have an inherent structure independent of other groupings, such as name groupings, locking groupings, etc., imposed on it. A data structure description consists of groupings of established data value type codes. The list of established data value types should be sufficient, through appropriate groupings, to describe any hierarchical structure of data.

The data type identifies how the data value is to be interpreted. A list of data type codes is given below. This list must be able to identify each data type that may exist in a data set in any machine in the network. However, for data sets that contain only data types of the machine on which it is stored, it is not necessary that a different code be defined for different forms of any single type that may exist among different machines. The data type specified in the description along with the identification of the machine at which the data is stored is sufficient to completely describe all such forms of the data types. A tentative list of machine dependent type codes, compiled by

G. Howe and T. Kibler, is as follows:

F floating point

I fixed

D double precision floating point

C character string

X complex

P packed decimal

L logical

It is desirable to be able to construct data sets that contain either data types not allowable at the machine at which the data set is stored or, possibly, even types that may not exist at any machine in the network. For example, one may wish to store eight bit data on a six bit machine. This may, in principle at least, be done by defining a logical data set containing eight bit bytes in terms of a real data set containing six bit characters. For this, however, data value type descriptors have to be defined that are machine independent. The basic machine independent data type is as follows:

B bit.

It is not clear at this time that any others are necessary since others can be built from this one. For convenience, other standard machine independent data types may later be defined.

Two other machine independent types are useful in describing structures. These are:

Z null

O omit.

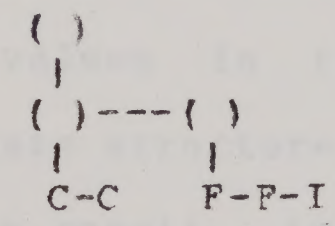
The null type indicates that there is no data corresponding

to this item; however, the item should be counted as existing in the structure. The omit type indicates just the opposite: there is data that should not be counted as an item, it should be ignored.

A grouping of data values is described by the list of elements of the grouping enclosed in parentheses. An element of a grouping may be either a data value as described by one of the data value type codes, or a grouping. The list consists of these elements, separated by commas and indicates that the elements appear in the grouping in the order indicated. For example, the description:

((C,C),(F,F,I))

describes a data collection consisting of two subgroupings, the first subgrouping consisting of two data values of type 'C', and the second subgrouping consisting of two data values of type 'F' followed by a data value of type 'I'. The structure of this data collection is thus a three level tree which may be shown in two dimensions as follows:



Properties

Other properties of data beyond that of the structure and composition of the data set have also to be described. These may be assigned to items of the data collection, where an item may be defined as an individual data value, or a grouping of these, by modifying the item description with the specification of the properties that apply to it. The

notation that will be used will be an infix notation of the form:

operand operator [[extent]]

where the operator indicates the property type, the operand the property value and the optional extent the number of items to which the property applies. Normally the property is assumed to apply to just the following item in the description. If the property is to apply to more than just the following item description, this is indicated by specifying a number as the extent, this number indicating how many of the following item definitions at the same level the property is to apply to.

Type - The structure description of the data set is a constitutional or syntactic description of the data set. In some cases it is necessary to give a description of the use or meaning of an element. For example, in some complex data structures, the linkages of the structure may be represented as data values in the data set. Thus, though the more complex data structure is represented in a hierarchical form and, as a result, is in a form describable by the above notation, the data values that represent the linkages, and their meaning, must somehow be represented in the data description in order for the complex data structure to be truly described. As another example, one may wish to ascribe to some level of the data structure the type 'record' so that the data set can be used by some system which uses the concept 'record' in accessing the data.

What an initial set of such types should be has not been decided.

Names - Items of a data structure can be given names by modifying the item description with a notation of the form:

name n ![extent]!.

Depending on the context of its use, the name can refer to the description itself or to the data pertaining to the named part of the description. The name is assumed to be unique only within the scope of extent of the next higher encompassing name unless otherwise indicated by giving another encompassing name as the scope. This may be the name of the whole data set or description, for example. The scope of a name is specified by preceding the inner name by the outer name or names, separated by dots. The name:

A.BETA

indicates that the scope of the name BETA is A.

The name applies to just the following item in the description unless otherwise indicated by including the extent parameter. For example, the description:

(An(C,C),(Bn[2]F,Cn[2]F,I))

indicates that name A is given to the item that contains two data values of type 'C', the name B to two data values, both of type 'F', and the name C to the last two data values, one of type 'F', and the other of type 'I'. Notice that with this notation, extents can overlap. For example, in the above description, the extent of name B overlapped that of name C.

In a description, the same name can be applied to more than one item definition either by use of the extent parameter, or by actually specifying the name at each item

to be included in the extent of the name. If a name is multiply-applied within the same scope, then the name is assumed to apply to the aggregate of the items to which it has been given. Thus it is possible to apply names to aggregates of items that are not necessarily sequential.

Lock - During the course of processing data, it may be necessary to lock out use of some portion of data to other users. Segmentation of a data set into units for locking purposes may be indicated by the notation:

k|[extent]|.

Whether or not the data is locked and the type of lock applied (for example, write protect, or read/write protect) is specified at the time the data is used.

Authorization - Authorization for a user to access data may be governed by some access code assigned to the data. This access code can be specified in the description by modifying the desired elements of the description with an indication of the code. The notation is:

code a |[extent]|

Control

Two modifiers are provided which govern the existence of items in the definition. The first is the repetition modifier:

factor r |[extent],|.

This causes the following item definition or item definitions (if the extent indicates more than one) to be

repeated. Thus the description

(3rC)

is equivalent to the description

(C,C,C).

The other control modifier is the condition modifier:

condition c [[extent]].

If the condition specified is not true, then the following item definition is ignored. The condition is specified by a Boolean expression.

Since several modifiers may apply to an item definition, there is a problem concerning the relationship among them. For example, if a repetition modifier and a conditional modifier apply to an item, does the condition apply to all the repeated items, or only to the first, assuming the extent of the condition modifier is one? The effect of multiple modifiers is dependent on the order in which they are evaluated. Two possible conventions come to mind. One says that repetitions are expanded first, then properties applied, and finally conditions applied to the resulting expanded item definitions, independent of the order in which the modifiers were specified in the description. Thus the description

(A=3c[4]4rF,I)

is equivalent to

(4rA=3c[4]F,I),

and, if the condition is true, is equivalent to

(F,F,F,F,I),

or, if the condition is not true, is equivalent to

(I).

The other convention is that the modifiers are evaluated in

the order in which they appear in the description, perhaps, in reverse order - the modifier immediately preceding an item definition is evaluated first, then the one next preceding, etc. This gives more flexibility of meaning to the multiple modifiers. For example, the descriptions

(A=3c3rC)

and

(3rA=3cC)

are not equivalent. In the first, only the first of the three repetitions is affected by the condition whereas in the second, all three repetitions are affected. Since this second convention is more flexible, it shall be the one assumed. This convention allows, for example, the repetition modifier to be applied to a named item as shown:

(3rAnC).

The name A applies to the three items (in effect, the name A is applied three times). This facility allows a name to be applied to a vertical column in a two dimensioned array by, for example, the description:

(3r[3]C,AnC,C)

which gives the name A to the second column of the 3x3 array.

Reference

Named descriptions, or parts of descriptions, that have already been defined may be inserted into a description using the notation:

$ specification.

In a description, the reference is used as an item

definition or a string of item definitions. The item definitions used are those defined by the name given. Names that apply to the named item or items as a whole in the description in which it is defined are ignored by the description at which is referred. However, names that apply to parts of the named item are carried over to the description at which it is referred. For example the description

(An(F,F),I,$A)

is equivalent to

(An(F,F),I,(F,F)).

Notice that the name "A" was not carried over to where the description was referenced since it applied to the referred-to item or items as a whole.

Parts of a data set or description must be able to be specified for use in a reference. This specification is in terms of the structure of the data set or description. The specification has the form of a data set name, or description name, followed by modifiers which particularize the specification to the part desired. The four types of modifiers are for going down a level, going up a level, going frontwards along a level, and going backwards.

Down - To go down a level from that previously specified, the modifier has the following form:

. item

or

. (item |,extent| |,=value|).

Having gone down a level, the item indicates which particular item at this level is the first (or only one)

desired. This may be a number or a name. If more than one are desired, then the extent indicates how many items. (* as extent means all remaining items at that level, ! means the first item that meets the conditions that may be set on it or subitems in following modifiers.) The items selected may be conditioned by their contents. If a value is given, then only those items with the value indicated are selected. For example,

A.1.1

specifies the first field of the first record of data set A,

A.(1,*).1

specified the first field of all the records of A,

A.1.(1,2)

specifies the first two fields of the first record of data set A,

A.(1,*).(1,="768174")

specified only the first fields of all the records of A that have value "768174", and

A.(1,!).(1,="768174")

finds the first field that has value "768174".

Up - To go up a level from that previously specified, the modifier has the following form:

' item

or

' (item |,extent| |,=value|).

Going up a level specifies the item up one level that contains the item previously specified. The item indicates which particular item at this level is desired where the containing item is considered the first. For example,

A.(1,!).(1,="768174")'1

specifies the first record whose first field has value "768174".

Forward. - To go forward on the same level as that previously specified, the modifier is as follows:

+ item

or

+ (item |,extent| |,=value|).

This modifier is useful when an item following the one which has a certain value in a field is desired. It may also be used when the data set name is really a pointer, into the data set, which has been set previously. Pointers may or may not be described in a section elsewhere.

Backward. - To go backward on the same level as that previously specified, the modifier has the following form:

- item

or

- (item |,extent| |,=value|).

An example of the use of this modifier is when an item preceding the one which has a certain value in a field is desired. This might be specified:

A.(1,!).(2,="768174")'1-1